Theoretische Informatik

FSS 2020 - Tutorium #4

Alexander Moch

20. Mai 2020

Aufgabe 4.1

Aufgabenstellung + Lösung

Zeige, dass für je zwei Sprachen $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ mit $L_1 \in \mathcal{P}$ und $\emptyset \neq L_2 \neq \Sigma_2^*$ gilt:

$L_1 \leq_p L_2$. Folgt aus $\mathcal{P} = \mathcal{NP}$, dass alle Sprachen aus $\mathcal{P}$ $\mathcal{NP}$-vollständig sind?

Lösung

Nach Definition von L_2 existieren (beliebig gewählte) $w_{yes} \in L_2$ und $w_{no} \in \Sigma_2^* \setminus L_2$.
Definiere $f : \Sigma_1^* \to \Sigma_2^*$ als:

$$f(w) = \begin{cases} w_{yes} & \text{falls } w \in L_1, \\ w_{no} & \text{sonst.} \end{cases}$$

Offensichtlich gilt für alle $w \in \Sigma_1^*$: $w \in L_1 \Leftrightarrow f(w) \in L_2$.

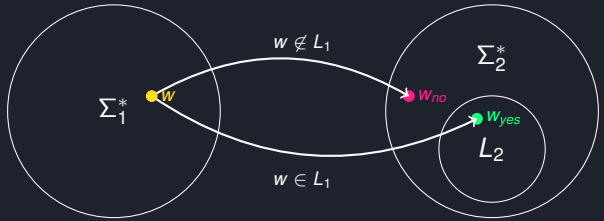
Wir zeigen nun, dass es einen Polynomialzeitalgorithmus M gibt, der f berechnet:

- Für eine gegebene Eingabe w entscheidet M, ob $w \in L_1$.
- Dies ist in polynomieller Zeit möglich, da $L_1 \in \mathcal{P}$.
- Dann berechnet M $f(w)$ gemäß der obigen Definition.

Lösung

Nach Definition von L_2 existieren (beliebig gewählte) $w_{yes} \in L_2$ und $w_{no} \in \Sigma_2^* \setminus L_2$.
Definiere $f : \Sigma_1^* \to \Sigma_2^*$ als:

$$f(w) = \begin{cases} w_{yes} & \text{falls } w \in L_1, \\ w_{no} & \text{sonst.} \end{cases}$$



$L_1 \in \mathcal{P}$.

Lösung - Fortsetzung

Nun folgt auch sofort, dass jede Sprache in

$$\{L \in \mathcal{P} | L \neq \emptyset \wedge \overline{L} \neq \emptyset\}$$

bezüglich $\leq_p$ $\mathcal{P}$-vollständig ist und damit, falls $\mathcal{P} = \mathcal{NP}$, $\mathcal{NP}$-vollständig.

Dies gilt aber nicht für L mit $\emptyset \in \{L, \overline{L}\}$, da es im Allgemeinen keine polynomielle Transformation f nach L geben kann, weil man die beiden notwendigen Fälle

- $f(w) \in L$ und
- $f(w) \notin L$

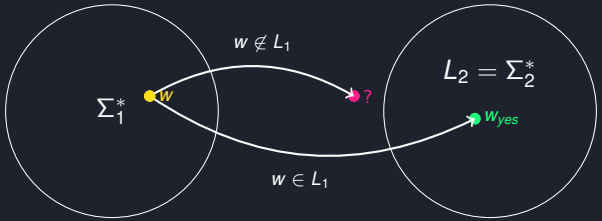
nicht zur Verfügung hat.

Lösung

Dies gilt aber nicht für L mit $\emptyset \in \{L, \overline{L}\}$, da es im Allgemeinen keine polynomielle Transformation f nach L geben kann, weil man die beiden notwendigen Fälle

- $f(w) \in L$ und
- $f(w) \notin L$

nicht zur Verfügung hat.

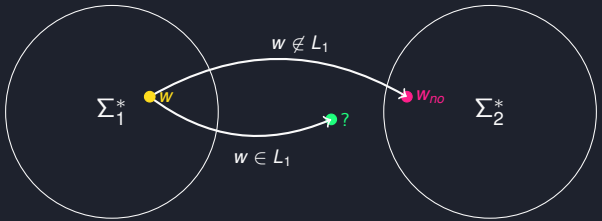


Lösung

Dies gilt aber nicht für L mit $\emptyset \in \{L, \overline{L}\}$, da es im Allgemeinen keine polynomielle Transformation f nach L geben kann, weil man die beiden notwendigen Fälle

- $f(w) \in L$ und
- $f(w) \notin L$

nicht zur Verfügung hat.



$$L_2 = \emptyset$$

Aufgabe 4.2

Aufgabe 4.2

Zeige, dass PARTITION $\in \mathcal{NPC}$ gilt, indem du eine polynomielle Transformation von RP* nach PARTITION angibst.

RP* (bzw. KP* im Englischen) bezeichnet das spezielle Rucksackproblem, dessen $\mathcal{NP}$-Vollständigkeit in der Vorlesung bewiesen wurde.

Aufgabe 4.2

Offensichtlich gilt: PARTITION $\in \mathcal{NP}$.[1]

In der Vorlesung haben wir gezeigt, dass ein spezielles Rucksackproblem RP* $\mathcal{NP}$-vollständig ist:

Gegeben $\vec{w} := (w_1, \ldots, w_n) \in \mathbb{N}^n$ und $W \in \mathbb{N}$:

Existiert $I \subseteq \{1, \ldots, n\}$, sodass $sum_{i \in I} w_i = W$?

[1] Einfache Übung: Wie sieht der entsprechende PRV-Algorithmus aus?

Aufgabe 4.2

Es genügt nun zu zeigen, dass RP* $\leq_p$ PARTITION gilt, indem wir eine entsprechende polynomielle Transformation f angeben.

Sei $(\vec{w}, W)$, mit $\vec{w} := (w_1, \ldots, w_n) \in \mathbb{N}^n$ und $W \in \mathbb{N}$, eine Eingabe für RP*.

Daraus konstruieren wir in polynomieller Zeit die Eingabe für PARTITION:

$f\big((\vec{w}, W)\big) := (w_1, \ldots, w_n, S - W + 1, W + 1)$, wobei $S := \sum_{i=1}^{n} w_i$.

Aufgabe 4.2

$(\vec{w}, W) \in \text{RP}^* \Rightarrow f\big((\vec{w}, W)\big) \in \text{PARTITION}$:

Falls I eine Lösung für das Problem RP* ist, erhalten wir mit $I \cup \{n+1\}$ eine Lösung für PARTITION, denn es gilt

$$\begin{aligned}\left(\sum_{i\in I} w_i\right) + (S - W + 1) &= W + S - W + 1 \\ &= S - W + W + 1 \\ &= \left(\sum_{i=1}^{n} w_i\right) - \left(\sum_{i\in I} w_i\right) + W + 1 \\ &= \left(\sum_{i\notin I} w_i\right) + (W + 1)\,.\end{aligned}$$

Aufgabe 4.2

$f\left((\vec{w}, W)\right) \in \text{PARTITION} \Rightarrow (\vec{w}, W) \in \text{RP}^*$:

- Die Summe aller Zahlen in der Eingabe für PARTITION beträgt $2S + 2$.
- Eine Lösung für PARTITION muss also so aussehen, dass jeder der beiden Teile sich zu $S + 1$ aufsummiert.
- Damit müssen die Zahlen $S - W + 1$ und $W + 1$ in verschiedenen Teilen sein.
- Die Zahlen, die $S - W + 1$ zu $S + 1$ ergänzen, haben die Summe W und bilden eine Lösung für die Eingabe von RP^*.

Aufgabe 4.3

Aufgabe 4.3

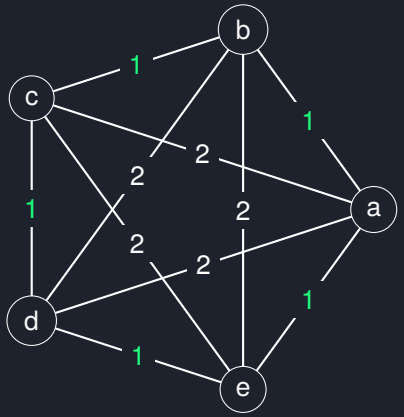
Zeige: Die Optimierungsvarianten von TSP und RP sind $\mathcal{NP}$-leicht.

Aufgabe 4.3

Beweis (TSP):

- Als Orakel aus $\mathcal{NP}$ benutzen wir die Entscheidungsvariante des TSP.
- Zunächst wollen wir mit Hilfe des Orakels die Kosten c_{opt} einer optimalen Rundreise berechnen.
- Es seien n die Anzahl der Knoten der Eingabe und c_{max} die Kosten der teuersten Kante.
- Dann ist $n \cdot c_{max}$ eine obere Schranke für die Kosten jeder möglichen Rundreise.
- Mit binärer Suche genügen $r = \lceil \log(n \cdot c_{max} + 1) \rceil$ Orakelfragen zur Berechnung von c_{opt}.
- Wir beachten dabei, dass r polynomiell in der Eingabelänge ist.

Beispiel: MinTSP $\leq_T$ TSP

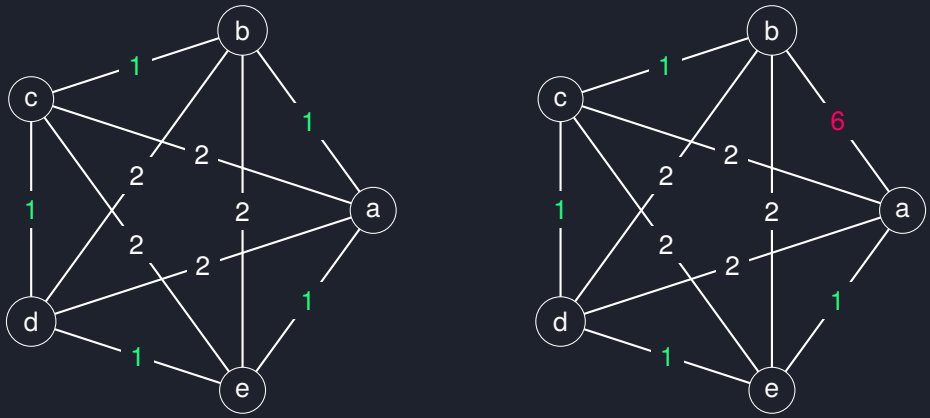


Gibt es eine Rundfahrt mit Kosten ≤ 10? $\overset{\mathcal{O}}{\to}$ Ja
Gibt es eine Rundfahrt mit Kosten ≤ 5? $\overset{\mathcal{O}}{\to}$ Ja
Gibt es eine Rundfahrt mit Kosten ≤ 3? $\overset{\mathcal{O}}{\to}$ Nein
Gibt es eine Rundfahrt mit Kosten ≤ 4? $\overset{\mathcal{O}}{\to}$ Nein
$\Longrightarrow$ Die optimale Rundfahrt hat Kosten von 5.

Aufgabe 4.3

- Danach genügen höchstens $n(n-1)$ weitere Orakelfragen zur Berechnung einer optimalen Rundreise.
- Dazu entfernen wir versuchsweise nacheinander alle Kanten, das heißt, wir ersetzen die Kosten durch $c_{opt} + 1$.
- Wenn nach einer Ersetzung keine Rundreise mit durch c_{opt} beschränkten Kosten existiert, ist die Kante für eine optimale Rundreise notwendig und wir setzen den Wert der Kante auf den alten Wert zurück.
- Andernfalls behalten wir den Kostenwert $c_{opt} + 1$ bei.
- Am Ende bilden alle Kanten, deren Kosten nicht auf $c_{opt} + 1$ gesetzt worden sind, eine optimale Rundreise.

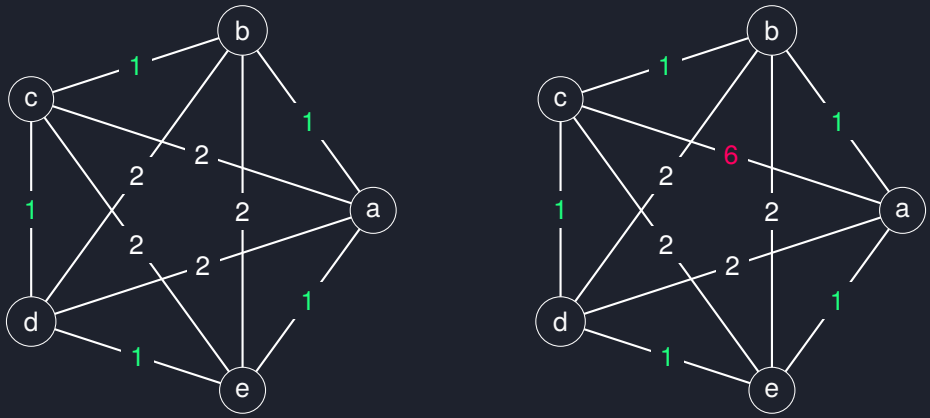
Beispiel: MinTSP $\leq_T$ TSP



Gibt es eine Rundfahrt mit Kosten 5? $\rightarrow$ Nein.

$\Longrightarrow$ Die Kante (a, b) ist Teil der optimalen Rundfahrt.

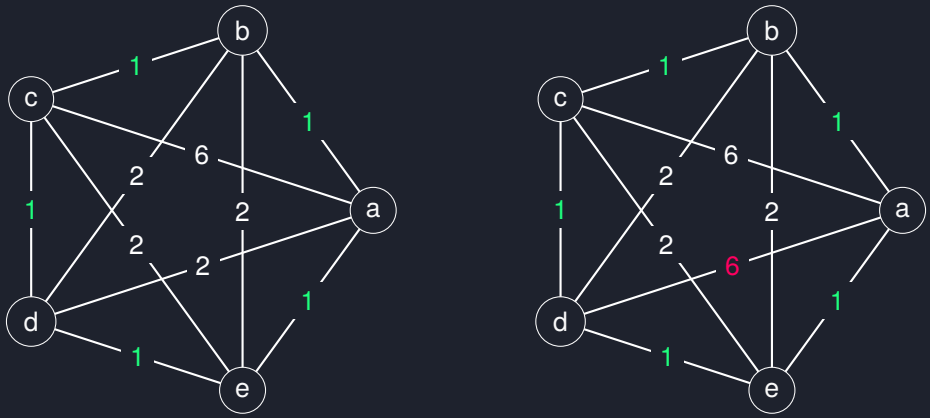
Beispiel: MinTSP $\leq_T$ TSP



Gibt es eine Rundfahrt mit Kosten 5? $\rightarrow$ Ja.

$\Longrightarrow$ Die Kante (a, c) ist *nicht* Teil der optimalen Rundfahrt.

Beispiel: MinTSP $\leq_T$ TSP



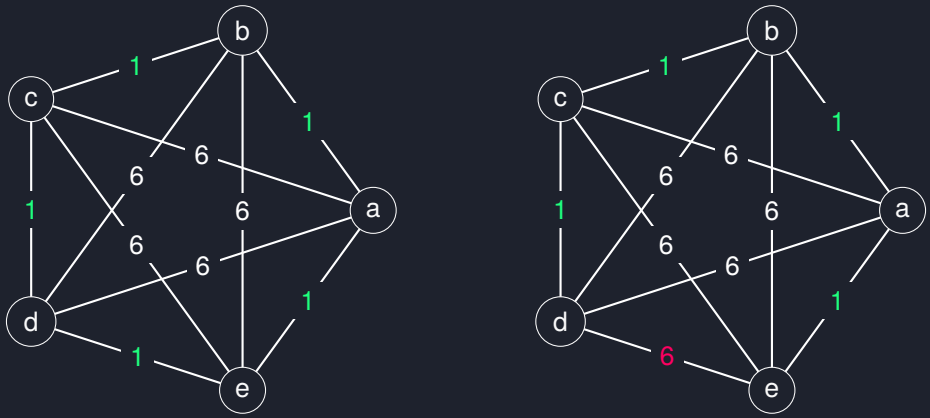
Gibt es eine Rundfahrt mit Kosten 5? $\rightarrow$ Ja.

$\Longrightarrow$ Die Kante (a, d) ist *nicht* Teil der optimalen Rundfahrt.

Beispiel: MinTSP $\leq_T$ TSP

...

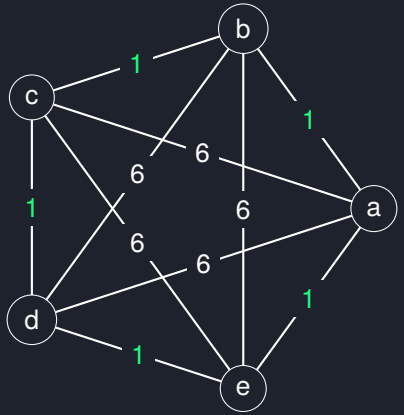
Beispiel: MinTSP $\leq_T$ TSP



Gibt es eine Rundfahrt mit Kosten 5? $\rightarrow$ Nein.

$\Longrightarrow$ Die Kante (d, e) ist Teil der optimalen Rundfahrt.

Beispiel: MinTSP $\leq_T$ TSP



Alle Kanten mit Gewicht $\neq 6$ sind Teil der optimalen Rundfahrt.

Lösung - RP

Beweis (RP):
Wir verwenden folgende Definitionen/Notationen:

- RP:
  - **Eingabe:** $X = (n, \vec{w}, \vec{c}, W, C)$ mit $n, W, C \in \mathbb{N}$, $\vec{w} = (w_1, \ldots, w_n) \in \mathbb{N}^n$, $\vec{c} = (c_1, \ldots, c_n) \in \mathbb{N}^n$.
  - **Akzeptiere** X g.d.w. $\exists I \subseteq \{1, \ldots, n\}$ mit $w(I) := \sum_{i \in I} w_i \leq W$ und $c(I) := \sum_{i \in I} c_i \geq C$.
- MaxRP:
  - **Eingabe:** $X = (n, \vec{w}, \vec{c}, W)$.
  - **Ausgabe:** $I^* \subseteq \{1, \ldots, n\}$ mit $w(I^*) \leq W$ und $c(I^*) = opt(X) := \max\{c(I) \mid I \subseteq \{1, \ldots, n\},\ w(I) \leq W\}$.

Lösung - RP

Polynomielle RP-OTM für MaxRP:

- Input: $X_{\text{MaxRP}} = (n, \vec{w}, \vec{c}, W)$
- Step 1: Compute $C^* := opt(X_{\text{MaxRP}})$ using oracle-based binary search in the interval $0, \ldots, \sum_{i=1}^{n} c_i$.
- Step 2:
  $X'_{\text{RP}} := (n, \vec{w}, \vec{c}, W, C^*)$
  for all $i = 1, \ldots, n$ **do**
    $X''_{\text{RP}} := X'_{\text{RP}} -$ Object i of X_{MaxRP}
    if $X''_{\text{RP}} \in$ RP **then**
      $X'_{\text{RP}} := X''_{\text{RP}}$
  output Index set $I^* \subseteq \{1, \ldots, n\}$ of all those objects of X_{MaxRP} finally remaining in X'_{RP}

Lösung - RP

Beachte: Anstatt die Objekte tatsächlich zu entfernen, hätten wir auch folgenden Ansatz (ähnlich wie im ersten Teil der Aufgabe) verwenden können:

(Alternative) Polynomielle RP-OTM für MaxRP:

- Input: $X_{\text{MaxRP}} = (n, \vec{w}, \vec{c}, W)$
- Step 1: Compute $C^* := opt(X_{\text{MaxRP}})$ using oracle-based binary search in the interval $0, \ldots, \sum_{i=1}^{n} c_i$.
- Step 2:
  $(w'_1, \ldots, w'_n) := \vec{w}$
  for all $i = 1, \ldots, n$ **do**
  if $(n, (w'_1, \ldots, w'_{i-1}, W+1, w'_{i+1}, \ldots, w'_n), \vec{c}, W, C^*) \in \text{RP}$ **then**
  $w'_i := W + 1$
  output $\{i \mid w'_i \neq W + 1\}$

Aufgabe 4.4

Das Bin Packing Problem

Bei dem BIN PACKING PROBLEM besteht jede Instanz aus

- einer Menge $A = \{a_1, \ldots, a_n\}$ von Objekten,
- einer Funktion $s : A \to \mathbb{N}$, die jedem a_i eine Größe zuordnet
- und einer Behältergröße $B \in \mathbb{N}$.

Bei der Optimierungsvariante fragt man, wie diese Objekte in möglichst wenige Behälter der Kapazität B gepackt werden können.

Genauer sucht man eine Partition $A_1 \dot\cup \cdots \dot\cup A_m = A$ mit m möglichst klein und $\max\{\sum_{a_i \in A_j} s(a_i) \mid 1 \leq j \leq m\} \leq B$.

Bei der Entscheidungsvariante ist für ein zusätzlich gegebenes $K \in \{1, \ldots, n\}$ zu entscheiden, ob K Behälter zum Verstauen aller Objekte ausreichen.

Aufgabe 4.4

Zeige:

- Die Entscheidungsvariante von BPP ist $\mathcal{NP}$-vollständig.

Lösung

Offensichtlich gehört dieses Problem, das wir mit DecBPP bezeichnen werden, zu $\mathcal{NP}$.

DecBPP ist nun $\mathcal{NP}$-vollständig, weil das $\mathcal{NP}$-vollständige Problem PARTITION ein Spezialfall davon ist.

Es gilt PARTITION $\leq_p$ DecBPP, da eine Eingabe $(p_1, \ldots, p_n) \in \mathbb{N}^n$ für PARTITION in Polynomialzeit in die Eingabe (A, s, B, K) mit

- $A := \{a_1, \ldots, a_n\}$,
- $s(a_i) := p_i$,
- $B := \frac{1}{2} \sum_{i=1}^{n} s(a_i)$,
- $K := 2$

für DecBPP transformiert werden kann.

Aufgabe 4.5

Aufgabe 4.5

Welche der folgenden Aussagen sind wahr?

a) $L_1 \leq_p L_2 \Rightarrow \overline{L}_1 \leq_p \overline{L}_2$,

b) $(L_1 \leq_p L_2 \ \wedge \ L_2 \in \mathcal{P}) \Rightarrow L_1 \in \mathcal{P}$,

c) $(L_1 \leq_p L_2 \ \wedge \ L_2 \in \mathcal{NP}) \Rightarrow L_1 \in \mathcal{NP}$,

d) $(L \in \mathcal{NP} \ \wedge \ L \notin \mathcal{P}) \Rightarrow \mathcal{P} \cap \mathcal{NPC} = \emptyset$.

---

a) Wahr: $(x \in L_1 \Leftrightarrow f(x) \in L_2) \Longleftrightarrow (x \notin L_1 \Leftrightarrow f(x) \notin L_2) = (x \in \overline{L}_1 \Leftrightarrow f(x) \in \overline{L}_2)$.

b) Wahr: Reduziere L_1 auf L_2 in Polynomialzeit. Dann entscheide L_2 in Polynomialzeit.

c) Wahr: Siehe Aufgabe 3.2.

d) Wahr: $(L \in \mathcal{NP} \ \wedge \ L \notin \mathcal{P})$ impliziert $\mathcal{P} \neq \mathcal{NP}$. In der Vorlesung wurde gezeigt:

$$\mathcal{P} \neq \mathcal{NP} \Rightarrow \mathcal{P} \cap \mathcal{NPC} = \emptyset.$$


That's all Folks!